**Cap4** **Clasificación con teoría de la probabilidad: Bayes naïve**

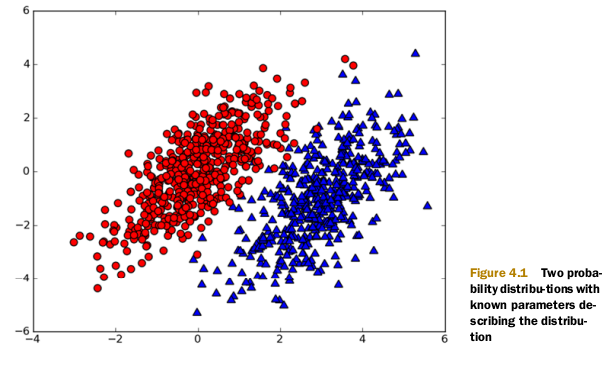
En los dos primeros capítulos pedimos a nuestro clasificador que tomara decisiones difíciles. Pedimos una respuesta definitiva para la pregunta "¿A qué clase pertenece esta instancia de datos?" A veces el clasificador obtuvo la respuesta incorrecta. Podríamos en cambio pedir al clasificador que nos diera una mejor estimación sobre la clase y asignar una estimación de probabilidad a esa mejor estimación.  
La teoría de la probabilidad constituye la base de muchos algoritmos de aprendizaje automático, por lo que es importante que obtenga una buena comprensión de este tema. Hemos tocado la probabilidad un poco en el capítulo 3 cuando estábamos calculando la probabilidad de que una característica tomara un valor dado

Calculamos la probabilidad contando el número de veces que la característica es igual a ese valor dividido por el número total de instancias en el conjunto de datos. Vamos a expandir un poco de allí en este capítulo. Veremos algunas maneras en que la teoría de la probabilidad puede ayudarnos a clasificar las cosas. Comenzamos con el clasificador probabilístico más simple y luego hacemos algunas suposiciones y aprendemos el ingenioso Bayes clasificador. Se llama ingenuo porque la formulación hace algunas suposiciones ingenuas. No te preocupes; Usted verá éstos en el detalle en un pedacito. Aprovecharemos plenamente las capacidades de procesamiento de texto de Python para dividir un documento en un vector de palabras.  
Esto se usará para clasificar el texto. Construiremos otro clasificador y veremos cómo lo hace en un conjunto de datos de correo basura del mundo real. Revisaremos la probabilidad condicional en caso de necesitar un refresco. Por último, mostraremos cómo puede poner lo que el clasificador ha aprendido en términos legibles desde un montón de anuncios de anuncios personales.

Clasificación con teoría bayesiana de la decisión

Pros: Trabaja con una pequeña cantidad de datos, maneja varias clases  
Contras: Sensible a cómo se preparan los datos de entrada  
Funciona con: Valores nominales

Naïve Bayes es un subconjunto de la teoría bayesiana de la decisión, por lo que necesitamos hablar sobre la teoría bayesiana de la decisión rápidamente antes de llegar a Bayes naïve.  
Supongamos por un momento que tenemos un conjunto de datos con dos clases de datos dentro. Una gráfica de estos datos se muestra en la figura 4.1.



Tenemos los datos mostrados en la figura 4.1 y tenemos un amigo que lee este libro; Encontró los parámetros estadísticos de las dos clases de datos. (No se preocupe sobre cómo encontrar los parámetros estadísticos para este tipo de datos ahora, vamos a llegar a que en el capítulo 10.) Tenemos una ecuación para la probabilidad de una pieza de datos pertenecientes a la Clase 1 (los círculos) : P1 (x, y), y tenemos una ecuación para la clase perteneciente a la Clase 2 (los triángulos): p2 (x, y).  
Para clasificar una nueva medida con características (x, y), usamos las siguientes reglas:

If p1(x, y) > p2(x, y), then the class is 1.

If p2(x, y) > p1(x, y), then the class is 2.

En pocas palabras, elegimos la clase con mayor probabilidad. Esa es la teoría bayesiana de la decisión en pocas palabras: elegir la decisión con la mayor probabilidad. Volvamos a los datos de la figura 4.1. Si puede representar los datos en seis números de punto flotante, y el código para calcular la probabilidad es de dos líneas en Python, ¿qué preferiría hacer?

1) Use kNN del capítulo 1 y haga 1.000 cálculos de distancia.  
2) Utilice los árboles de decisión del capítulo 2 y haga una división de los datos una vez a lo largo del eje x y una vez a lo largo del eje y.  
3) Calcular la probabilidad de cada clase y compararla

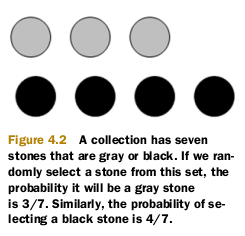
El árbol de decisiones no sería muy exitoso, y kNN requeriría muchos cálculos en comparación con el cálculo de probabilidad simple. Dado este problema, la mejor opción sería la comparación de probabilidad que acabamos de discutir.  
Vamos a tener que ampliar las medidas de probabilidad p1 y p1 que proporcioné aquí. Para poder calcular p1 y p2, debemos discutir la probabilidad condicional. Si cree que tiene un buen manejo de la probabilidad condicional, puede omitir la siguiente sección.

¿Bayes?

Esta interpretación de la probabilidad que usamos pertenece a la categoría llamada probabilidad bayesiana; Es popular y funciona bien. La probabilidad bayesiana se nombra después de Thomas Bayes, que era un teólogo del siglo XVIII. La probabilidad bayesiana permite que el conocimiento y la lógica anteriores se apliquen a declaraciones inciertas. Hay otra interpretación llamada probabilidad de frecuencia, que sólo saca conclusiones de los datos y no permite la lógica y el conocimiento previo.

Conditional probability

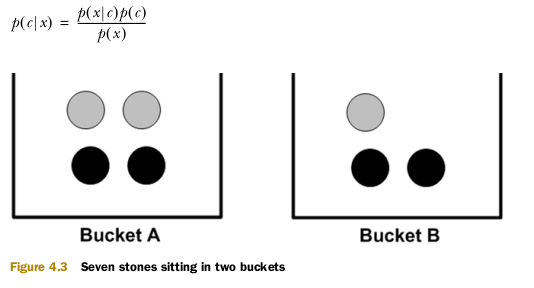
Vamos a pasar unos minutos hablando de probabilidad y probabilidad condicional. Si está cómodo con el símbolo p (x, y | c1), puede omitir esta sección.  
Supongamos por un momento que tenemos un frasco que contiene siete piedras. Tres de estas piedras son grises y cuatro son negras, como se muestra en la figura 4.2. Si metemos una mano en este frasco y tiramos aleatoriamente una piedra, ¿cuáles son las probabilidades de que la piedra sea gris? Hay siete piedras posibles y tres son de color gris, por lo que la probabilidad es 3/7. Cuál es el

Probabilidad de agarrar una piedra negra? Es 4/7. Escribimos la probabilidad de gris como P (gris). Calculamos la probabilidad de dibujar una piedra gris P (gris) contando el número de piedras grises y dividiendo esto por el número total de piedras.  
¿Y si las siete piedras estuvieran en dos baldes?  
Figura 4.2 Una colección tiene siete  
Esto se muestra en la figura 4.3.

Si desea calcular el P (gris) o P (negro), ¿saber el cambio de cubo de la respuesta? Si quería calcular la probabilidad de dibujar una piedra gris del cubo B, probablemente podría averiguar cómo hacerlo. Esto se conoce como probabilidad condicional. Estamos calculando la probabilidad de una piedra gris, dado que la piedra desconocida proviene del cubo B. Podemos escribir esto como P (gray | bucketB), y esto sería leído como "la probabilidad de que el cubo dado en gris B." No es difícil ver que P (gray | bucketA) es 2/4 y P (gray | bucketB) es 1/3.  
Para formalizar la forma de calcular la probabilidad condicional, podemos decir

P(gray|bucketB) = P(gray and bucketB)/P(bucketB)

Vamos a ver si eso tiene sentido: P (gris y bucketB) = 1/7. Esto se calculó tomando el número de piedras grises en el cubo B y dividiendo por el número total de piedras. Ahora, P (cubo B) es 3/7 porque hay tres piedras en el cubo B del total de siete piedras.  
Por último, P (gray | bucketB) = P (gris y bucketB) / P (bucketB) = (1/7) / (3/7) = 1/3.  
Esta definición formal puede parecer demasiado trabajo para este ejemplo simple, pero será útil cuando tengamos más funciones. También es útil tener esta definición formal si alguna vez necesitamos manipular algebraicamente la probabilidad condicional.  
Otra forma útil de manipular probabilidades condicionales se conoce como la regla de Bayes.  
La regla de Bayes nos dice cómo intercambiar los símbolos en una declaración de probabilidad condicional. Si tenemos P (x | c) pero queremos tener P (c | x), podemos encontrarlo con lo siguiente:



Ahora que hemos discutido la probabilidad condicional, necesitamos ver cómo aplicar esto a nuestro clasificador. La siguiente sección discutirá cómo usar las probabilidades condicionales con la teoría bayesiana de la decisión.

Clasificación con probabilidades condicionales

En la sección 4.1, dije que la teoría bayesiana de la decisión nos dijo que encontraramos las dos probabilidades:

If p1(x, y) > p2(x, y), then the class is 1.

If p2(x, y) > p1(x, y), then the class is 2.

Estas dos reglas no cuentan toda la historia. Acabo de dejarlos como p1 () y p2 () para mantenerlo tan simple como sea posible. Lo que realmente necesitamos comparar son p (c1 | x, y) y p (c2 | x, y).  
Vamos a leer estos para enfatizar lo que significan. Dado un punto identificado como x, y, ¿cuál es la probabilidad de que vino de la clase c1? ¿Cuál es la probabilidad de que venga de la clase c2 ?.  
El problema es que la ecuación de nuestro amigo es p (x, y | c1), que no es lo mismo.  
Podemos usar la regla de Bayes para cambiar las cosas. La regla de Bayes se aplica a estas declaraciones como sigue:



Con estas definiciones, podemos definir la regla de clasificación bayesiana:



Usando la regla de Bayes, podemos calcular este desconocido a partir de tres cantidades conocidas. Pronto escribiremos un código para calcular estas probabilidades y clasificar los elementos usando la regla de Bayes.  
Ahora que hemos introducido un poco de teoría de la probabilidad, y usted ha visto cómo usted puede construir un clasificador con él, vamos a poner esto en la acción. La siguiente sección introducirá una aplicación simple pero poderosa del clasificador bayesiano.

Clasificación de documentos con Bayes naïve

Una aplicación importante del aprendizaje automático es la clasificación automática de documentos.  
En la clasificación de documentos, todo el documento como un correo electrónico individual es nuestra instancia y las características son las cosas en ese correo electrónico. El correo electrónico es un ejemplo que sigue apareciendo, pero se pueden clasificar historias de noticias, discusiones de tablones de mensajes, presentaciones con el gobierno o cualquier tipo de texto. Usted puede mirar los documentos por las palabras usadas en ellos y tratar la presencia o ausencia de cada palabra como característica. Esto le daría tantas características como hay palabras en su vocabulario. Naïve Bayes -una extensión del clasificador bayesiano introducido en la última sección- es un algoritmo popular para el problema de clasificación de documentos.  
Anteriormente mencioné que vamos a usar palabras individuales como características y buscar la presencia o ausencia de cada palabra. ¿Cuántas características es esa? ¿Qué lenguaje (humano) estamos asumiendo? Puede ser más de un idioma.

El número total estimado de palabras en el idioma inglés es más de 500,000.1] Para poder leer en inglés, se estima que usted necesita entender miles de palabras.  
Supongamos que nuestro vocabulario tiene 1.000 palabras. Para generar buenas distribuciones de probabilidad, necesitamos suficientes muestras de datos. Llamemos a esto N muestras. En ejemplos anteriores en este libro, teníamos 1.000 ejemplos para el sitio de citas, 200 ejemplos por dígito en el reconocimiento de escritura y 24 ejemplos para nuestro árbol de decisiones.  
Tener 24 ejemplos era un poco bajo, 200 muestras eran mejores, y 1.000 muestras era grande. En el ejemplo de citas teníamos tres características. Estadísticas nos dice que si necesitamos N muestras para una característica, necesitamos N10 para 10 características y N1000 para nuestro vocabulario de 1.000 características. El número será muy grande muy rápidamente.

Si asumimos la independencia entre las características, entonces nuestros puntos de datos N1000 se reducen a 1000 \* N. Por independencia quiero decir independencia estadística; Una característica o palabra es tan probable por sí misma como está al lado de otras palabras. Estamos asumiendo que la palabra bacon es tan probable que aparezca junto a insalubres ya que está al lado de delicioso. Sabemos que esta suposición no es verdadera; Tocino casi siempre aparece cerca de delicioso, pero muy rara vez cerca de insalubres. Esto es lo que se entiende por ingenuo en el ingenuo Bayes clasificador. La otra suposición que hacemos es que cada característica es igualmente importante. Sabemos que tampoco es cierto.  
Si estábamos tratando de clasificar un mensaje de la junta de publicación como inadecuado, probablemente no es necesario mirar a 1.000 palabras; Tal vez 10 o 20 lo harán. A pesar de las pequeñas fallas de estos supuestos, ingenuo Bayes funciona bien en la práctica.  
En este punto ya sabes lo suficiente sobre este tema para empezar con algún código. Si todo no tiene sentido en este momento, podría ayudar a ver esto en acción. En la siguiente sección, comenzaremos a implementar el naïve Bayes clasificador en Python. Vamos a pasar por todo lo que se necesita para clasificar el texto con Python.

**Enfoque general para Bayes naïve  
1.** Recopilar: Cualquier método. Usaremos feeds RSS en este capítulo.  
2. Prepare: Se necesitan valores numéricos o booleanos.  
3. Analizar: Con muchas características, trazar características no es útil. Mirar los histogramas es una mejor idea.  
4. Tren: Calcular las probabilidades condicionales de las características independientes.  
5. Prueba: Calcule la tasa de error.  
6. Uso: Una aplicación común de naïve Bayes es la clasificación de documentos. Puede utilizar Bayes naïve en cualquier clasificación. No tiene que ser texto.

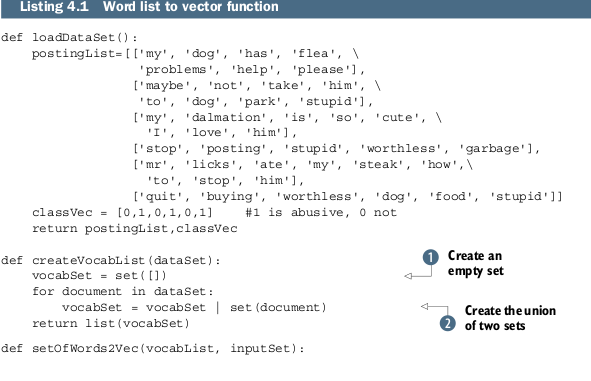
Clasificación de texto con Python

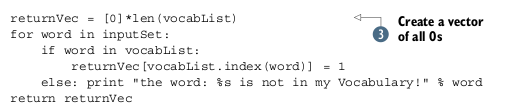
Para obtener las características de nuestro texto, tenemos que dividir el texto. Pero cómo hacemos eso? Nuestras características van a ser fichas que obtenemos del texto. Un token es cualquier combinación de caracteres. Puede pensar en símbolos como palabras, pero podemos usar cosas que no son palabras como direcciones URL, direcciones IP o cualquier cadena de caracteres. Reduciremos cada fragmento de texto a un vector de fichas donde 1 representa el token existente en el documento y 0 representa que no está presente.  
Para ver esto en acción, hagamos un filtro rápido para un tablero de mensajes en línea que señale un mensaje como inapropiado si el autor usa lenguaje negativo o abusivo. Filtrar este tipo de cosas es común porque las publicaciones abusivas hacen que la gente no vuelva y pueda perjudicar a una comunidad en línea. Tendremos dos categorías: abusivas y no. Usaremos 1 para representar abusivo y 0 para representar no abusivo.

Primero, vamos a mostrar cómo transformar listas de texto en un vector de números.  
A continuación, mostraremos cómo calcular las probabilidades condicionales de estos vectores. Luego, crearemos un clasificador y, finalmente, veremos algunas consideraciones prácticas para implementar Bayes naïve en Python.

Preparar: crear vectores de texto a partir del texto

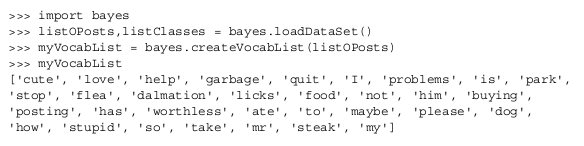
Vamos a empezar a mirar el texto en forma de vectores de palabras o vectores de token, es decir, transformar una oración en un vector. Consideramos todas las palabras de todos nuestros documentos y decidimos qué utilizaremos para un vocabulario o conjunto de palabras que consideraremos. Después, necesitamos transformar cada documento individual en un vector de nuestro vocabulario. Para empezar, abra su editor de texto, cree un nuevo archivo llamado bayes.py y agregue el código de la siguiente lista.

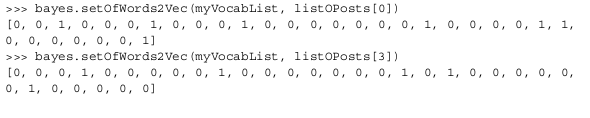




La primera función crea algunos datos de ejemplo con los que experimentar. La primera variable devuelta de loadDatSet () es un conjunto tokenized de documentos de un tablero de mensaje de los amantes de Dalmatian (manchado de la raza del perro). El texto se ha dividido en un juego de fichas. La puntuación se ha eliminado de este texto también. Regresaremos al proceso de texto más tarde. La segunda variable de loadDatSet () devuelve un conjunto de etiquetas de clase. Aquí tienes dos clases, abusivas y no abusivas. El texto ha sido etiquetado por un ser humano y se usará para entrenar un programa para detectar automáticamente mensajes abusivos.

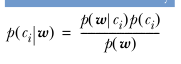
A continuación, la función createVocabList () creará una lista de todas las palabras únicas en todos nuestros documentos. Para crear esta lista única, utilice el tipo de datos establecido en Python. Puede dar una lista de elementos al constructor de conjuntos y sólo devolverá una lista única. En primer lugar, crear un conjunto vacío. B A continuación, agrega el conjunto con un nuevo conjunto de cada documento.  
C El | Operador se utiliza para la unión de dos conjuntos; Recuerde que este es el operador OR bit a bit de C. OR binario y la unión de conjunto también utilizan los mismos símbolos en la notación matemática.  
Finalmente, después de tener nuestra lista de vocabulario, puede usar la función setOfWords2Vec (), que toma la lista de vocabulario y un documento y genera un vector de 1s y 0s para representar si una palabra de nuestro vocabulario está presente o no en el documento dado . A continuación, crear un vector de la misma longitud que la lista de vocabulario y llenarlo con 0s. D A continuación, pasa por las palabras del documento, y si la palabra está en la lista de vocabulario, establece su valor en 1 en el vector de salida. Si todo va bien, no deberías probar si una palabra está en vocablist, pero puedes usarla más tarde.  
Ahora veamos estas funciones en acción. Guarde bayes.py e introduzca lo siguiente en su shell de Python:

Si examina esta lista, verá que no hay palabras repetidas. La lista no está clasificada, y si desea ordenarla, puede hacerlo más tarde.  
Veamos la siguiente función setOfWords2Vec ():

Esto ha tomado nuestra lista de vocabulario o lista de todas las palabras que le gustaría examinar y creó una función para cada uno de ellos. Ahora, cuando se aplica un documento determinado (una publicación en el sitio dálmata), se transformará en un vector de palabras. Compruebe si esto tiene sentido. ¿Cuál es la palabra en el índice 2 en myVocabList? Debe ser ayuda. Esta palabra debe estar en nuestro primer documento. Ahora compruebe que no está en nuestro cuarto documento.

Tren: cálculo de probabilidades a partir de vectores de palabras

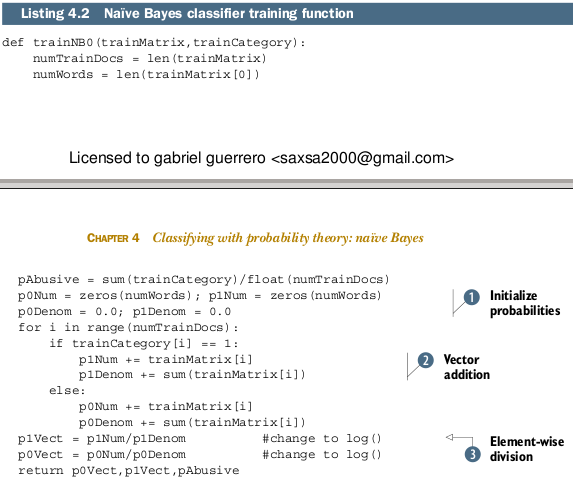
Ahora que has visto cómo convertir de palabras a números, veamos cómo calcular las probabilidades con estos números. Usted sabe si una palabra ocurre en un documento, y usted sabe a qué clase pertenece el documento. ¿Recuerdas la regla de Bayes de la sección 3.2? Ha sido reescrito aquí, pero he cambiado las x, y a w. El tipo en negrita significa que es un vector; Es decir, tenemos muchos valores, en nuestro caso, tantos valores como palabras en nuestro vocabulario.



Vamos a usar el lado derecho de la fórmula para obtener el valor de la izquierda. Haremos esto para cada clase y compararemos las dos probabilidades. ¿Cómo conseguimos las cosas a la derecha? Podemos calcular p (ci) sumando cuántas veces vemos la clase i (posts abusivos o posts no abusivos) y luego dividiendo por el número total de posts. ¿Cómo podemos obtener p (w | ci)? Aquí es donde entra nuestra suposición ingenua. Si expandimos w en características individuales, podríamos reescribir esto como p (w0, w1, w2..wN | ci). Nuestra suposición de que todas las palabras eran independientes, y algo llamado independencia condicional, dice que podemos calcular esta probabilidad como p (w0 | ci) p (w1 | ci) p (w2 | ci) ... p (wN | ci) .  
Esto hace nuestros cálculos mucho más fáciles.

Pseudocódigo para esta función sería así:  
*Cuente el número de documentos en cada clase  
Para cada documento de formación:  
   Para cada clase:  
Si aparece un token en el documento ➞ incrementar el recuento para ese token  
Incrementar el recuento de fichas  
Para cada clase:  
Para cada ficha:  
Dividir el recuento de tokens por el total de tokens para obtener probabilidades condicionales devolver probabilidades condicionales para cada clase*

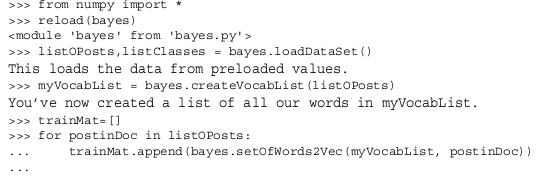
El código en la siguiente lista hará estos cálculos para nosotros. Abra su editor de texto e inserte este código en bayes.py. Esta función utiliza algunas funciones de NumPy, así que asegúrese de agregar de numpy import \* a la parte superior de bayes.py.



La función en el listado 4.2 toma una matriz de documentos, trainMatrix, y un vector con las etiquetas de clase para cada uno de los documentos, trainCategory. Lo primero que haces es calcular la probabilidad de que el documento sea un documento abusivo (clase = 1). Esto es P (1) de arriba; Porque este es un problema de dos clases, puede obtener P (0) por 1-P (1). Para más de un problema de dos clases, tendría que modificar esto un poco.  
Inicializa el numerador y el denominador para los cálculos p (wi | c1) y p (wi | c0). B Ya que tienes tantos ws, vas a usar matrices NumPy para calcular estos valores rápidamente. El numerador es una matriz NumPy con el mismo número de elementos que tiene palabras en su vocabulario. En el bucle for se realizan bucle sobre todos los documentos en trainMatrix, o nuestro conjunto de entrenamiento. Cada vez que aparece una palabra en un documento, se incrementa el recuento para esa palabra (p1Num o p0Num) y el número total de palabras de un documento se resume en todos los documentos. C Haga esto para ambas clases.

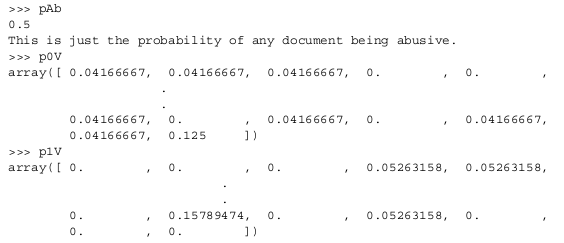
Por último, se divide cada elemento por el número total de palabras para esa clase. 3  
Esto se hace de forma compacta en NumPy dividiendo una matriz por un flotador. Esto no se puede hacer con listas regulares de Python. Pruébalo para verlo por ti mismo. Finalmente, se devuelven los dos vectores y una probabilidad.

Vamos a probar esto. Después de agregar el código del listado 4.2 a bayes.py, abra su shell de Python e introduzca lo siguiente:

Este bucle rellena la lista trainMat con vectores de palabra. Ahora vamos a obtener las probabilidades de ser abusivo y los dos vectores de probabilidad:



Miremos dentro de cada una de estas variables:



Primero, encontró la probabilidad de que un documento fuera abusivo: pAb; Esto es 0.5, que es correcto. A continuación, encontró las probabilidades de las palabras de nuestro vocabulario dada la clase de documento. Veamos si esto tiene sentido. La primera palabra en nuestro vocabulario es linda. Aparece una vez en la clase 0 y nunca en la clase 1. Las probabilidades son 0,04166667 y 0,0. Esto tiene sentido. Vamos a buscar la mayor probabilidad.  
Eso es 0.15789474 en la matriz P (1) en el índice 21. Si miras la palabra en myVocabList en el índice 26, verás que es la palabra estúpido. Esto le dice que la palabra estúpido es más indicativo de una clase 1 (abusivo).  
Antes de que podamos ir a la clasificación con esto, tenemos que abordar algunos defectos en la función anterior.

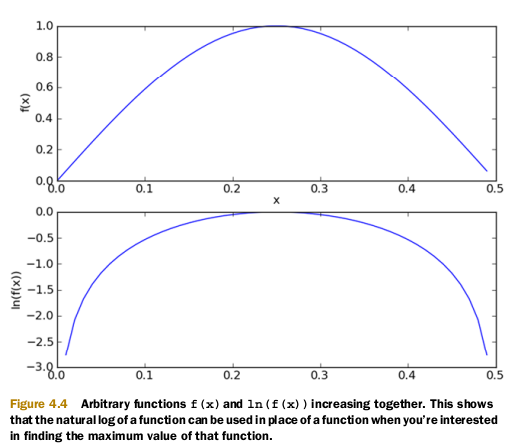
Test: modificando el clasificador para las condiciones del mundo real

Cuando intentamos clasificar un documento, multiplicamos muchas probabilidades para obtener la probabilidad de que un documento pertenezca a una clase dada. Esto se verá como p (w0 | 1) p (w1 | 1) p (w2 | 1). Si alguno de estos números es 0, entonces cuando los multiplicamos juntos obtendremos 0. Para disminuir el impacto de esto, inicializaremos todos los conteos de ocurrencia a 1, e inicializaremos los denominadores a 2.  
Abra bayes.py en el editor de texto y cambie las líneas 4 y 5 de trainNB0 () a

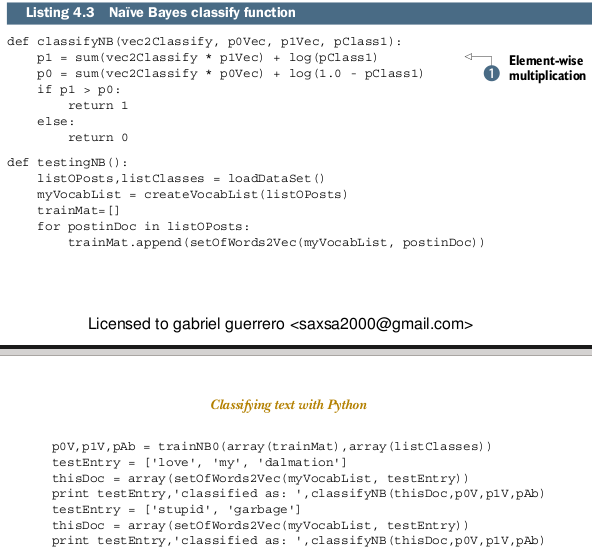


Otro problema es underflow: hacer demasiadas multiplicaciones de números pequeños.  
Cuando vamos a calcular el producto p (w0 | ci) p (w1 | ci) p (w2 | ci) ... p (wN | ci) y muchos de estos números son muy pequeños, Una respuesta incorrecta. (Trate de multiplicar muchos números pequeños en Python, eventualmente redondea a 0.) Una solución a esto es tomar el logaritmo natural de este producto. Si recuerda del álgebra, ln (a \* b) = ln (a) + ln (b). Hacer esto nos permite evitar el problema de error de desbordamiento o redondeo. ¿Perdemos algo usando el registro natural de un número en lugar del número en sí? La respuesta es no. La figura 4.4 representa dos funciones, f (x) e ln (f (x)). Si examina estas dos parcelas, verá que aumentan y disminuyen en las mismas áreas, y tienen sus picos en las mismas áreas. Sus valores son diferentes, pero está bien. Para modificar nuestro clasificador para tener en cuenta esto, modifique las dos últimas líneas antes de la devolución para que se vea así:

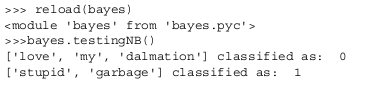




Ahora estamos listos para construir el clasificador completo. Es bastante simple cuando estamos usando matemáticas vectoriales con NumPy. Abra el editor de texto y agregue el código de la siguiente lista a bayes.py.



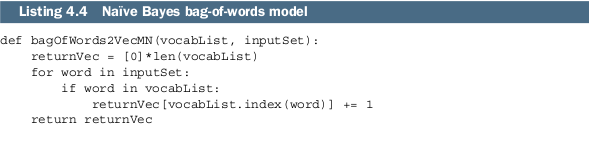
El código del listado 4.3 toma cuatro entradas: un vector para clasificar llamado vec2Classify y tres probabilidades calculadas en la función trainNB0 (). Utilice matrices NumPy para multiplicar dos vectores. B La multiplicación es element-wise; Es decir, se multiplican los primeros elementos de ambos vectores, luego los segundos elementos, y así sucesivamente. Luego agrega los valores de todas las palabras de nuestro vocabulario y agrega esto a la probabilidad de registro de la clase. Finalmente, verá qué probabilidad es mayor y devolverá la etiqueta de la clase. Eso no es demasiado difícil, ¿verdad?  
La segunda función en el listado 4.3 es una función de conveniencia para terminar todo correctamente y ahorrarle tiempo de escribir todo el código de la sección 4.3.1.  
Vamos a probarlo. Después de agregar el código de la lista 4.3, ingrese lo siguiente en su shell de Python:



Cambie el texto y vea lo que el clasificador escupe. Este ejemplo es demasiado simplista, pero demuestra cómo funciona el ingenuo Bayes clasificador. Seguidamente haremos algunos cambios para que funcione aún mejor.

Preparar: el modelo de documento de bolsa de palabras

Hasta este punto hemos tratado la presencia o ausencia de una palabra como una característica. Esto podría ser descrito como un modelo de conjunto de palabras. Si una palabra aparece más de una vez en un documento, podría transmitir algún tipo de información sobre el documento sobre la palabra que aparece en el documento o no. Este enfoque se conoce como un modelo de bolsa de palabras. Una bolsa de palabras puede tener múltiples ocurrencias de cada palabra, mientras que un conjunto de palabras puede tener sólo una aparición de cada palabra. Para acomodar esto necesitamos cambiar ligeramente la función setOfWords2Vec () y llamarla bagOfWords2VecMN ().  
El código para usar el modelo de bolsa de palabras se da en el siguiente listado. Es casi idéntico a la función setOfWords2Vec () enumerada anteriormente, excepto que cada vez que se encuentra con una palabra, incrementa el vector de palabras en lugar de establecer el vector de palabra en 1 para un índice dado.

Ahora que tenemos un clasificador construido, deberíamos ser capaces de poner esto en la acción de clasificación de spam.

Ejemplo: clasificar correo spam con Bayes naïve

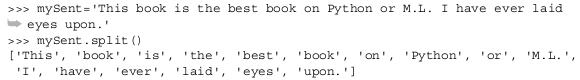
En el ejemplo simple anterior se importó una lista de cadenas. Para usar Bayes naïve en algunos problemas de la vida real, tendremos que poder pasar de un cuerpo de texto a una lista de cadenas y luego un vector de palabras. En este ejemplo vamos a visitar el famoso uso de Bayes naïve: filtrado de spam de correo electrónico. Veamos primero cómo abordaremos este problema con nuestro marco general.

*Ejemplo: utilizar Bayes naïve para clasificar el correo electrónico  
1. Recopilar: Archivos de texto proporcionados.  
2. Preparar: Analizar texto en vectores de token.  
3. Analice: Inspeccione los tokens para asegurarse de que el análisis se realizó correctamente.  
4. Tren: Use trainNB0 () que creamos anteriormente.  
5. Prueba: Utilice classifyNB () y cree una nueva función de prueba para calcular la tasa de error sobre un conjunto de documentos.  
6. Utilice: Construya un programa completo que clasifique un grupo de documentos e imprima documentos mal clasificados en la pantalla.*

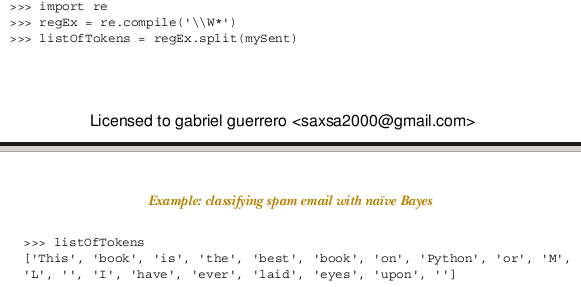
Primero, crearemos algún código para analizar texto en fichas. A continuación escribiremos una función que relaciona el análisis sintáctico y el código de clasificación anteriormente de este capítulo. Esta función también probará el clasificador y nos dará una tasa de error.

Preparar: texto tokenizing

La sección anterior mostró cómo crear vectores de palabras y utilizar bayes naïve para clasificar con estos vectores de palabras. La palabra vectores en la sección anterior vino premade. Veamos cómo crear sus propias listas de palabras a partir de documentos de texto.  
Si tiene una cadena de texto, puede dividirla utilizando el método .split () de cadena Python.  
Veamos esto en acción. Ingrese lo siguiente en su shell de Python:



Eso funciona bien, pero la puntuación se considera parte de la palabra. Puede utilizar expresiones regulares para dividir la oración en cualquier cosa que no sea una palabra o un número:



Ahora tienes una lista de palabras. Pero tienes algunas cuerdas vacías de las que necesitas deshacerte.  
Puede contar la longitud de cada cadena y devolver sólo los elementos mayores que 0.



Finalmente, la primera palabra de la oración está en mayúscula. Si estuviera mirando frases, esto sería útil. Estás mirando una bolsa de palabras, así que quieres que todas las palabras se vean iguales si están en el medio, el final o el comienzo de una oración.  
Python tiene métodos integrados para convertir cadenas a todas las minúsculas (.lower ()) o todas las mayúsculas (.upper ()). Esto resolverá nuestro problema. Cambiemos nuestra comprensión de lista a lo siguiente:

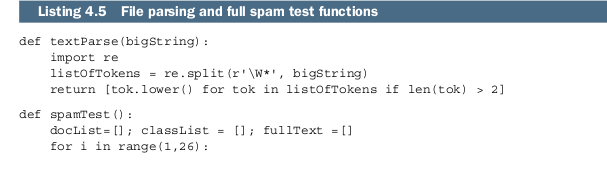


Ahora vamos a ver esto en acción con un correo electrónico completo de nuestro conjunto de datos de correo electrónico. El conjunto de datos de correo electrónico está en una carpeta llamada correo electrónico, con dos subcarpetas llamadas spam y jamón.



El archivo llamado 6.txt en la carpeta de jamones es bastante largo. Es de una compañía que me dice que ya no apoyan algo. Una cosa a notar es que ahora tenemos palabras como en y py porque originalmente formaban parte de una URL: /answer.py?hl=es&answer=174623. Cuando dividimos la URL recibimos muchas palabras. Nos gustaría deshacernos de estas palabras, así que filtraremos palabras con menos de tres caracteres.  
Utilizamos una regla general de análisis de texto para este ejemplo. En un programa de análisis del mundo real, debería tener filtros más avanzados que busquen cosas como HTML y URIs.  
En este momento, un URI terminará como una de nuestras palabras; Www.whitehouse.gov terminará como tres palabras. El análisis de texto puede ser un proceso involucrado. Vamos a crear una función de huesos, y se puede modificar como mejor le parezca.

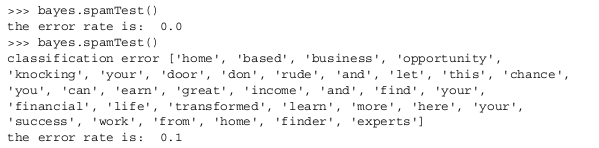
Prueba: validación cruzada con Bayes naïve



La primera función, textParse (), toma una cadena grande y analiza el texto en una lista de cadenas. Elimina cualquier cosa bajo dos caracteres de largo y convierte todo a minúsculas. Hay mucho más análisis que se puede hacer en esta función, pero es lo suficientemente bueno para nuestros propósitos.

La segunda función, spamTest (), automatiza el naïve Bayes clasificador de spam. Carga los archivos de texto de spam y de jamón en listas de palabras. B A continuación, crea un conjunto de prueba y un conjunto de entrenamiento. Los correos electrónicos que entran en el conjunto de pruebas y el conjunto de entrenamiento serán seleccionados al azar. En este ejemplo, tenemos 50 correos electrónicos totales (no muchos). Diez de los correos electrónicos se seleccionan al azar para ser utilizados en el conjunto de pruebas. Las probabilidades se calcularán a partir de sólo los documentos del conjunto de entrenamiento. La variable trainingSet de Python es una lista de números enteros de 0 a 49. A continuación, selecciona aleatoriamente 10 de esos archivos. C Cuando se selecciona un número, se agrega al conjunto de pruebas y se elimina del conjunto de entrenamiento. Esta selección al azar de una parte de nuestros datos para el conjunto de entrenamiento y una porción para el conjunto de prueba se llama validación cruzada hold-out. Usted ha hecho solamente una iteración, pero para obtener una buena estimación del verdadero error de nuestro clasificador, debe hacer esto varias veces y tomar la tasa de error promedio.

El siguiente para el bucle itera a través de todos los elementos del conjunto de prueba y crea vectores de palabras de las palabras de cada correo electrónico y el vocabulario usando setOfWords2Vec (). Estas palabras se usan en traindNB0 () para calcular las probabilidades necesarias para la clasificación.  
A continuación, iterar a través del conjunto de pruebas y clasificar cada correo electrónico en el conjunto de prueba. D Si el correo electrónico no está clasificado correctamente, se incrementa el recuento de errores y, finalmente, se informa del porcentaje total de error.



Prueba esto. Después de haber introducido el código del listado 4.5, ingrese lo siguiente en su shell de Python:

La función spamTest () muestra la tasa de error de 10 correos electrónicos seleccionados al azar.  
Dado que estos son seleccionados al azar, los resultados pueden ser diferentes cada vez. Si hay un error, mostrará la lista de palabras de ese documento para darle una idea de lo que fue clasificado erróneamente. Para obtener una buena estimación de la tasa de error, debe repetir este procedimiento varias veces, digamos 10, y el promedio de los resultados. Lo hice y conseguí una tasa de error promedio del 6%.  
El error que sigue apareciendo es una pieza de spam que se clasificó erróneamente como jamón. Es mejor que un pedazo de spam se escabulle a través del filtro que un correo electrónico válido que se mete en la carpeta de correo no deseado. Hay maneras de sesgar al clasificador para no cometer estos errores, y hablaremos de éstos en el capítulo 7.

Ahora que hemos utilizado Bayes naïve para clasificar los documentos, vamos a ver otro uso para ello. El siguiente ejemplo mostrará cómo interpretar los conocimientos adquiridos de la formación del ingenioso Bayes clasificador.

Ejemplo: usar Bayes naïve para revelar actitudes locales a partir de anuncios personales

Nuestro próximo y último ejemplo es divertido. Examinamos dos aplicaciones prácticas del naïve Bayes clasificador. El primero era filtrar mensajes maliciosos en un sitio web, y el segundo era filtrar el spam en el correo electrónico. Hay una serie de otros usos para la clasificación. He visto a alguien tomar el ingenuo Bayes clasificador y entrenarlo con perfiles de redes sociales de las mujeres que le gustaban y las mujeres que no le gustaba y luego usar el clasificador para probar cómo le gustaría a una persona desconocida. La gama de posibilidades está limitada sólo por su imaginación. Se ha demostrado que cuanto más viejo es, mejor se convierte su vocabulario. ¿Podríamos adivinar la edad de una persona por las palabras que usan? ¿Podríamos adivinar otros factores sobre la persona? A los anunciantes les encantaría conocer datos demográficos específicos sobre una persona para orientar mejor los productos que promueven. ¿De dónde sacarías ese material de capacitación? El Internet abunda con material de capacitación. Casi todos los nichos imaginables tienen una comunidad dedicada donde las personas se han identificado como pertenecientes a esa comunidad. El sitio de los dálmatas utilizado en la sección 4.3.1 es un gran ejemplo.

En este último ejemplo, tomaremos algunos datos de anuncios personales de varias personas para dos ciudades diferentes en los Estados Unidos. Vamos a ver si las personas en diferentes ciudades utilizan palabras diferentes. Si lo hacen, ¿cuáles son las palabras que utilizan? ¿Pueden las palabras que usan las personas darnos alguna idea de lo que es importante para las personas en diferentes ciudades?

*Ejemplo: usar bayes naïve para encontrar palabras usadas localmente  
1. Recopilar: Recopilar de los feeds RSS. Necesitaremos construir una interfaz para los feeds RSS.  
2. Preparar: Analizar texto en vectores de token.  
3. Analice: Inspeccione los tokens para asegurarse de que el análisis se realizó correctamente.  
4. Tren: Use trainNB0 () que creamos anteriormente.  
5. Prueba: Vamos a ver la tasa de error para asegurarse de que esto realmente está funcionando. Podemos hacer modificaciones en el tokenizer para mejorar la tasa de error y los resultados.  
6. Uso: Construiremos un programa completo para envolver todo juntos. Exhibirá las palabras más comunes dadas en dos alimentaciones de RSS.*

Vamos a utilizar la ciudad de la que proviene cada anuncio para entrenar a un clasificador y luego ver cómo bien lo hace. Finalmente, no vamos a usar esto para clasificar nada. Vamos a ver las palabras y las puntuaciones de probabilidad condicional para ver si podemos aprender algo específico de una ciudad sobre otra.

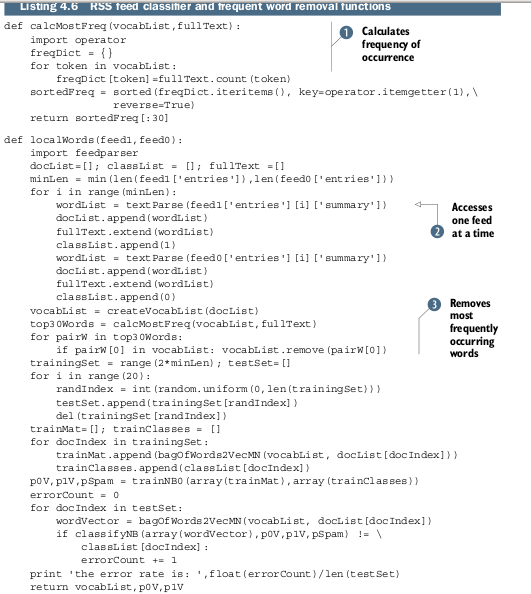
Recopilar: importar fuentes RSS

Lo primero que vamos a hacer es usar Python para descargar el texto. Por suerte, el texto está disponible en forma RSS. Ahora todo lo que necesitamos es un lector de RSS. Universal  
Feed Parser es la librería RSS más común para Python.  
Puede ver la documentación aquí: http://code.google.com/p/feedparser/. Deberías poder instalarlo como otros paquetes de Python, descomprimiendo el paquete descargado, cambiando tu directorio al paquete descomprimido y escribiendo >> python setup.py install en el símbolo del sistema.  
Vamos a usar los anuncios personales de Craigslist, y espero que podamos cumplir con los términos del servicio. Para abrir el feed RSS de Craigslist, ingrese lo siguiente en su shell de Python:



He decidido utilizar el paso, o estrictamente platónico, sección de Craigslist porque otras secciones pueden obtener un poco de lascivia. Usted puede jugar con la alimentación y echa un vistazo a la gran documentación en feedparser.org. Para acceder a una lista de todas las entradas tipo

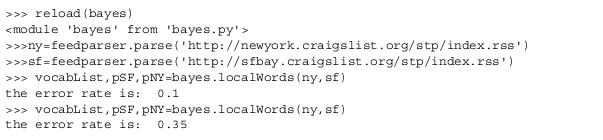
Puede crear una función similar a spamTest () para automatizar las pruebas. Abra su editor de texto e ingrese el código de la siguiente lista.



El código en la lista 4.6 es similar a la función spamTest () en el listado 4.5 con algunas características añadidas. Una función auxiliar está incluida en la lista 4.6; La función se llama calcMostFreq (). B La función de ayuda pasa por cada palabra del vocabulario y cuenta cuántas veces aparece en el texto. El diccionario se clasifica por frecuencia de mayor a menor, y las 100 palabras superiores se devuelven. Verás por qué esto es importante en un segundo.

La siguiente función, localWords (), toma dos feeds como argumentos. Las alimentaciones deben cargarse fuera de esta función. La razón para hacerlo es que los feeds pueden cambiar con el tiempo, y si desea realizar algunos cambios en nuestro código para ver cómo se realiza, debe tener los mismos datos de entrada. Recargar los feeds le dará nuevos datos, y no estará seguro de si nuestro código ha cambiado o de nuevos datos han cambiado nuestros resultados. La función localWords () es mayormente la misma que spamTest () del listado 4.5. Las diferencias son que se accede a los feeds C en lugar de a los archivos, y se llama a calcMostFreq () para obtener las 100 palabras principales y luego se eliminan estas palabras. D El resto de la función es similar a spamTest (), excepto que la última línea devuelve valores que utilizará posteriormente.

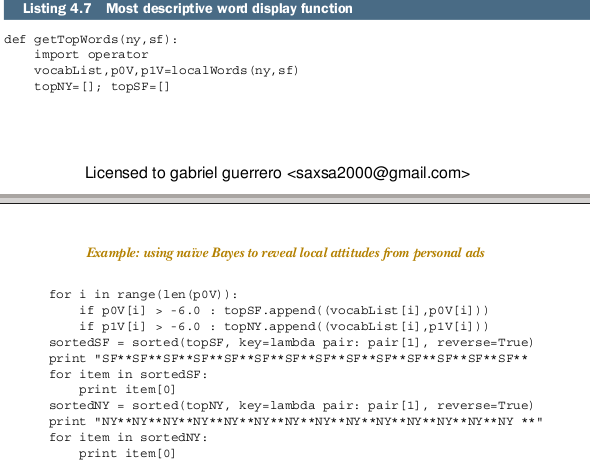
Puede comentar las tres líneas que eliminaron las palabras más utilizadas y ver el rendimiento antes y después. D Cuando lo hice, tuve una tasa de error del 54% sin estas líneas y un 70% con las líneas incluidas. Una observación interesante es que las 30 palabras superiores en estos postes componen cerca del 30% de todas las palabras usadas. El tamaño de la vocabList era ~ 3000 palabras cuando estaba probando esto. Un pequeño porcentaje del total de palabras forma una gran parte del texto. La razón de esto es que un gran porcentaje del lenguaje es redundancia y pegamento estructural. Otro enfoque común es no sólo eliminar las palabras más comunes, sino también eliminar este pegamento estructural de una lista predefinida. Esto se conoce como una lista de palabra parada, y hay una serie de fuentes de esta disponible. (En el momento de escribir esto, http://www.ranks.nl/resources/stopwords.html tiene una buena lista de palabras en varios idiomas.)  
Después de introducir el código desde la lista 4.6 en bayes.py, puede probarlo en Python escribiendo lo siguiente:



Para obtener una buena estimación de la tasa de error, debe hacer múltiples ensayos de este y tomar el promedio. La tasa de error aquí es mucho mayor que para las pruebas de spam. Eso no es un problema enorme porque estamos interesados en las probabilidades palabra, no realmente clasificar nada. Usted puede jugar alrededor del número de palabras quitado por caclMostFreq () y ver cómo la tasa de error cambia.

Analizar: mostrar palabras usadas localmente

Puede ordenar los vectores pSF y pNY y luego imprimir las palabras de vocabList en el mismo índice. Hay un último código que hace esto por ti. Abra bayes.py una vez más e ingrese el código del siguiente listado.



La función getTopWords () en el listado 4.7 toma los dos feeds y los primeros trenes y prueba el naïve clasificador de Bayes. Se devuelven las probabilidades utilizadas. A continuación, crea dos listas y almacena tuplas dentro de las listas. En lugar de simplemente devolver las primeras palabras X,  
Devolver todas las palabras por encima de un determinado umbral. Las tuplas son ordenadas por sus probabilidades condicionales.  
Para ver esto en acción, ingrese lo siguiente en su shell de Python después de haber guardado bayes.py....



Las palabras de esta salida son entretenidas. Una cosa a tener en cuenta: un montón de stop wordsappear en la salida. Sería interesante ver cómo cambiarían las cosas si  
Se eliminaron las palabras de parada fijas. En mi experiencia, el error de clasificación también bajará.

Resumen

El uso de probabilidades a veces puede ser más efectivo que el uso de reglas duras para la clasificación. La probabilidad bayesiana y la regla de Bayes nos da una forma de estimar probabilidades desconocidas a partir de valores conocidos.  
Puede reducir la necesidad de una gran cantidad de datos asumiendo la independencia condicional entre las características de sus datos. La suposición que hacemos es que la probabilidad de una palabra no depende de ninguna otra palabra en el documento. Sabemos que esta suposición es un poco simple. Es por eso que se conoce como naïve Bayes. A pesar de sus suposiciones incorrectas, Bayes naïve es eficaz en la clasificación.

Hay varias consideraciones prácticas al implementar Bayes naïve en un lenguaje de programación moderno. Underflow es un problema que se puede resolver usando el logaritmo de probabilidades en sus cálculos. El modelo de bolsa de palabras es una mejora en el modelo de conjunto de palabras al aproximarse a la clasificación de documentos. Hay una serie de otras mejoras, como la eliminación de palabras de parada, y puede pasar mucho tiempo optimizando un tokenizer.  
La teoría de la probabilidad que aprendió en este capítulo se utilizará de nuevo más adelante en el libro, y este capítulo fue una gran introducción al poder completo de la teoría de la probabilidad bayesiana. Vamos a tomar un descanso de la teoría de la probabilidad. A continuación veremos un método de clasificación llamado regresión logística y algunos algoritmos de optimización.